

FIG. 1

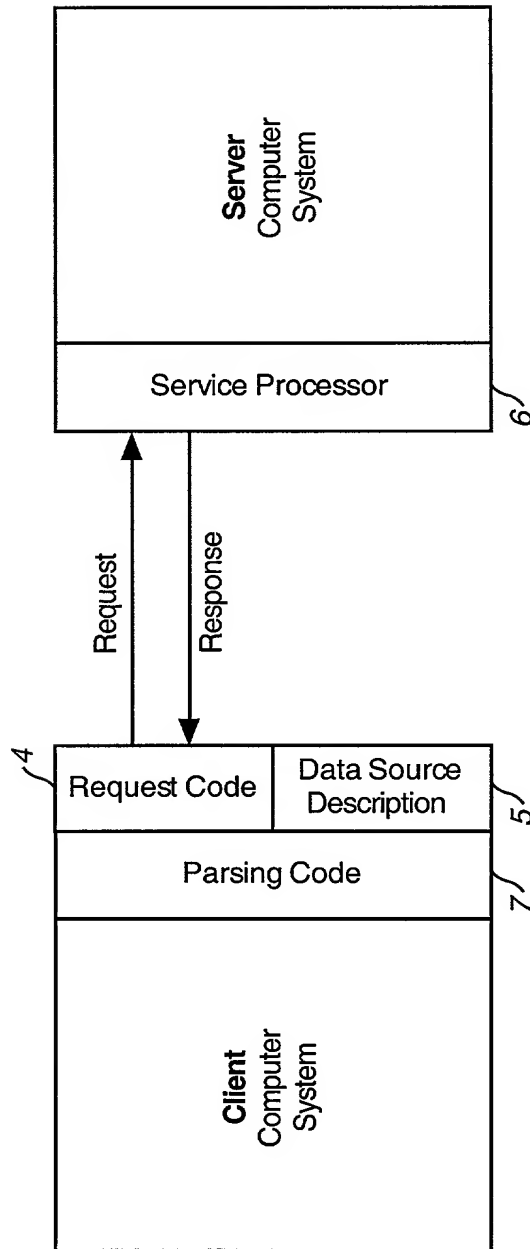


FIG. 2

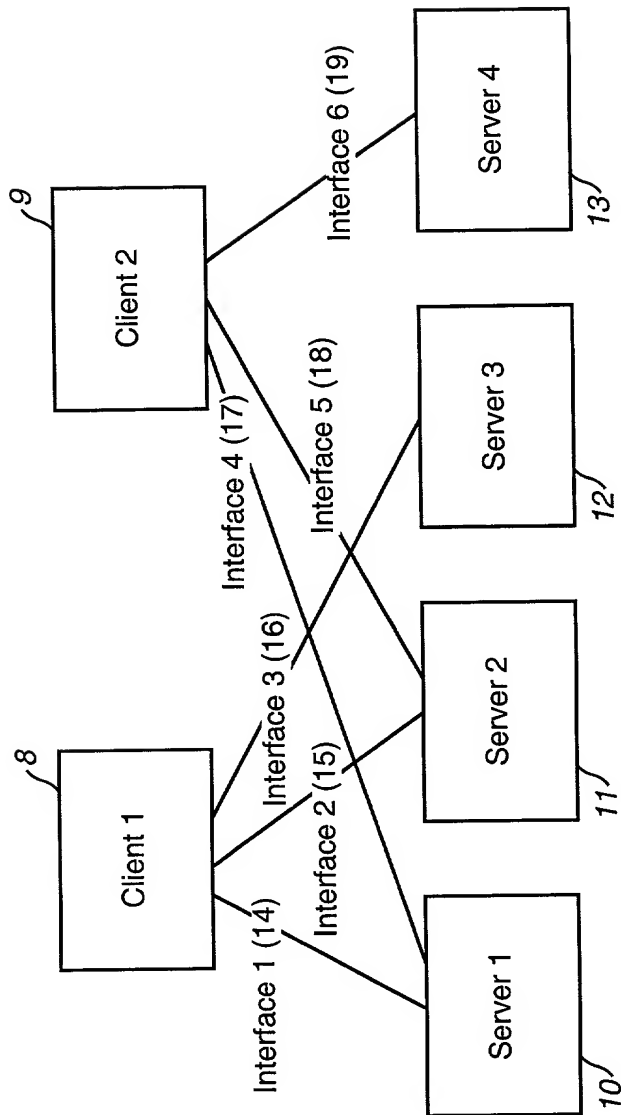


FIG. 3

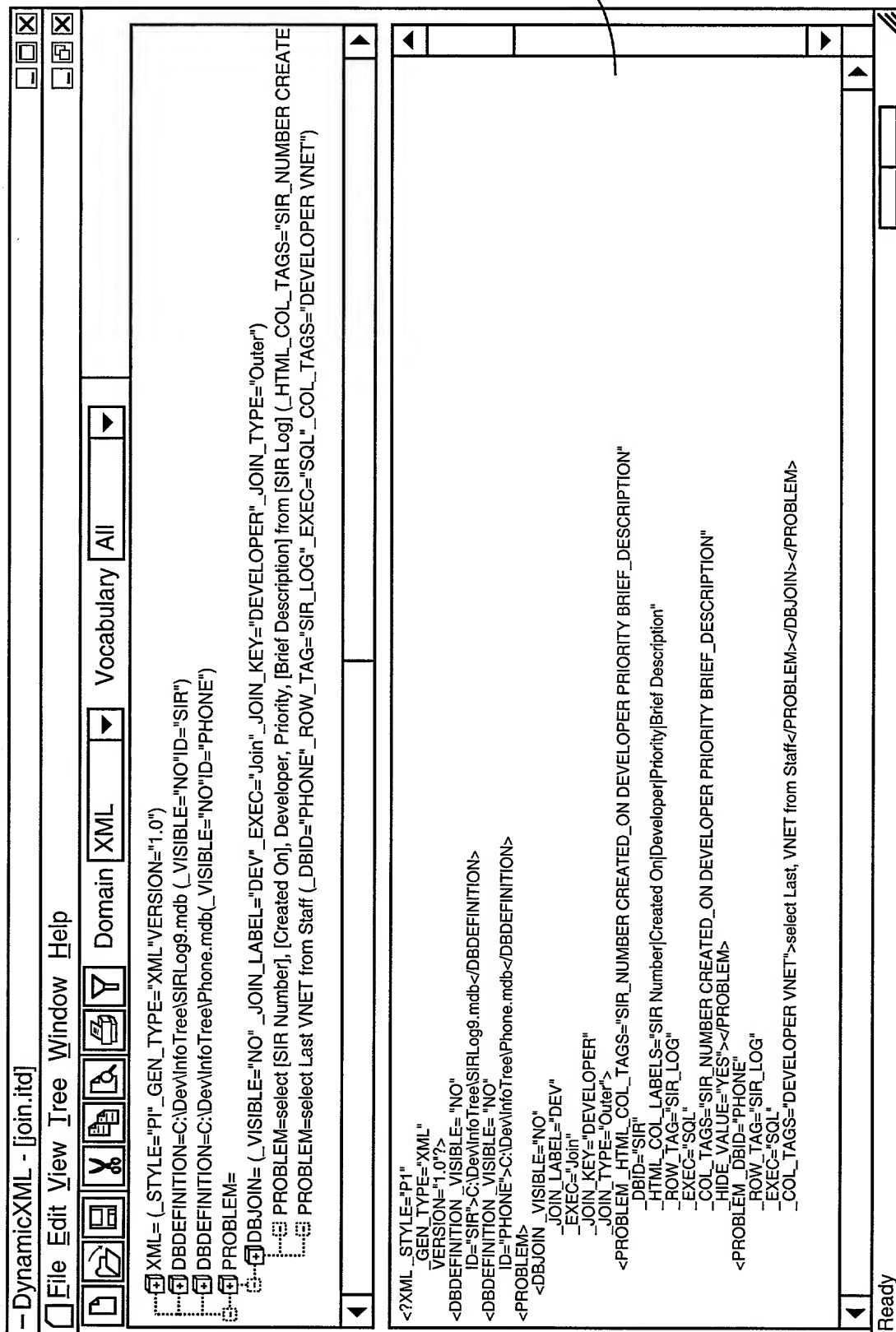


FIG. 4a

DynamicXML - [join.itd]

File Edit View Tree Window Help

Domain XML Vocabulary All

XML= (_STYLE="PI" _GEN_TYPE="XML" _VERSION="1.0")

DBDEFINITION=C:\Dev\InfoTree\SIRLog9.mdb (_VISIBLE="NO" ID="SIR")

DBDEFINITION=C:\Dev\InfoTree\Phone.mdb (_VISIBLE="NO" ID="PHONE")

PROBLEM=

DBJOIN= (_VISIBLE="NO" _JOIN_LABEL="Join" _JOIN_KEY="DEVELOPER" _JOIN_TYPE="Outer")

PROBLEM=select [SIR Number], [Created On], Developer, Priority, [Brief Description] from [SIR Log] (_HTML_COL_TAGS="SIR_NUMBER CREATE

PROBLEM=select Last VNET from Staff (_DBID="PHONE" _ROW_TAG="SIR_LOG" _EXEC="SQL" _COL_TAGS="DEVELOPER VNET")

<?XML VERSION="1.0"?>

<PROBLEM>

<DEV>

<SIR_NUMBER>7</SIR_NUMBER>

<CREATED_ON>10/27/97 2:31:44 PM</CREATED_ON>

<DEVELOPER>Joy</DEVELOPER>

<PRIORITY>2</PRIORITY>

<BRIEF_DESCRIPTION> 'Middle Name' is inconsistently labeled throughout; 'Mid. Name', 'Middle', and 'MI'. When the customer populates any of the date field

<VNET>555-3786</VNET></DEV>

<DEV>

<SIR_NUMBER>9</SIR_NUMBER>

<CREATED_ON>10/27/97 2:34:10 PM</CREATED_ON>

<DEVELOPER>Joy</DEVELOPER>

<PRIORITY>1</PRIORITY>

<BRIEF_DESCRIPTION>The HELP functionality is not present throughout the application -- We are unable to test the help functionality due to this limitation

<VNET>555-3786</VNET></DEV>

<DEV>

<SIR_NUMBER>25</SIR_NUMBER>

<CREATED_ON>10/29/97 3:31:29 PM</CREATED_ON>

<DEVELOPER>Zon</DEVELOPER>

<PRIORITY>2</PRIORITY>

<BRIEF_DESCRIPTION>Retested 12/10/97 (Tim Donegan) Created a NYNEX Resale/Local Line product AMI (555)247-7777 I was able to select each of the following

Original SIR: User able to select mutually exclusive options, i.e. Call Waiting and Call Waiting Deluxe, Caller ID and Caller ID w/name. </BRIEF_DESCRIPTION>

<VNET>555-6096</VNET></DEV>

<DEV>

<SIR_NUMBER>56</SIR_NUMBER>

<CREATED_ON>11/3/97 1:44:07 PM</CREATED_ON>

<DEVELOPER>Tazowski</DEVELOPER>

<PRIORITY>3</PRIORITY>

<BRIEF_DESCRIPTION>PIN Field has no delimiter. UPDATE 11/25: The field still accepts alpha characters</BRIEF_DESCRIPTION>

<VNET>555-5780</VNET></DEV>

<DEV>

<SIR_NUMBER>108</SIR_NUMBER>

Ready

Ready

FIG. 4b

Tree Item Information

Style:

▼

Element

Item:

▼

Problem

Type:

▼

PROBLEM

Settings

Execution Type:

▼

SQL

☐ Invisible

SQL...

OK

Cancel

Value:

File...

select [SIR Number], [Created On], Developer, Priority, [Brief Description] from [SIR Log]

Attributes:

Name	Value
<div>HTML_COL_TAGS</div>	SIR_NUMBER CREATED_ON DEVE...
<div>DBID</div>	SIR
<div>HTML_COL_LABE...</div>	SIR Number[Created On Developer Pr...
<div>ROW_TAG</div>	SIR_LOG
<div>COL_TAGS</div>	SIR_NUMBER CREATED_ON DEVE...
<div>HIDE_VALUE</div>	YES

New

Edit

Delete

Preview

FIG. 4c

Select ODBC Tables

Database: C:\Dev\InfoTree\SIRLog9.mdb

Table: SIR Log

Table Columns:

Notify Date
Primary Reason
Priority
Problem Area
Project Release
Reproductibility
Requested By
Resolution
Responsibility
S_Screen
Screen

>>>

Remove

Up

Down

Add All

Clear All

Selected Columns:

SIR Number
Created On
Developer
Priority
Brief Description

select [SIR Number], [Created On], Developer, Priority, [Brief Description] from [SIR Log]

Attributes

XML

Column Tags

Row Tag

HTML Table

Column Labels

Column Tags

OK

Cancel

26

FIG. 4d

Figure 5. System Parameter File page 1

```

1      <?XML VERSION="1.0"?>
2      <GENERAL>
3          <DOMAINS>
4              <DOMAIN NAME="XML">
5                  <_STYLE KEY="ELEM"
6                      LABEL="Element"><%T%A%V%C%/_STYLE>
7                  <_STYLE KEY="PI"
8                      LABEL="Processing Instruction"><?%T%V%A?></_STYLE>
9                  <_STYLE KEY="COMMENT"
10                     LABEL="Comment"><!-- %V --></_STYLE>
11                  <_STYLE KEY="TEXT"
12                     LABEL="Text">%V</_STYLE>
13                  <_STYLE KEY="CDATA"
14                     LABEL="CDATA"><![CDATA[%V]]></_STYLE>
15                  <EMPTY EMPTY_STYLES="ELEM"><%T%A%></EMPTY>
16                  <HEADER>
17                      <?xml version="1.0"?></HEADER>
18                  <EXTENSION SYSTEM="c:\dev\agentview\Release\AgentView"
19                      LABEL="View"/></DOMAIN>
20              <DOMAIN NAME="Key-Value">
21                  <_STYLE KEY="ELEM"
22                      LABEL="Element">%T="%V"%C</_STYLE>
23                  <HEADER>
24                      <DOCTYPE /></HEADER></DOMAIN></DOMAINS>
25          <EXEC TYPES>
26              <EXEC_TYPE KEY="SQL"
27                  LABEL="SQL"/>
28              <EXEC_TYPE KEY="ADO"
29                  LABEL="ADO"/>
30              <EXEC_TYPE KEY="SHELL"
31                  LABEL="Shell"/>
32              <EXEC_TYPE KEY="JOIN"
33                  LABEL="Join"/></EXEC_TYPES></GENERAL>
34          <DEFINITIONS>
35              <DEFAULT OUTPUT_FONT_SIZE>24</DEFAULT_OUTPUT_FONT_SIZE>
36              <DEFAULT_OUTPUT_FONT>Courier New</DEFAULT_OUTPUT_FONT>
37              <ATTR_COL_LABEL_SEP>|</ATTR_COL_LABEL_SEP>
38              <ATTR_EXEC>EXEC</ATTR_EXEC>
39              <SPLIT_HORIZ>1</SPLIT_HORIZ>
40              <NORMALIZE_NAME_REPLACE_CHARS>./$</NORMALIZE_NAME_REPLACE_CHARS>
41              <NORMALIZE_NAME_MAKE_UPPER>0</NORMALIZE_NAME_MAKE_UPPER>
42              <XML_CHAR_MAP><=[>=</XML_CHAR_MAP>
43              <TREE_VIEW_FORMAT>Type %T, Attrs: %A, Value=%V</TREE_VIEW_FORMAT></DEFINITIONS>
44          <VOCABULARIES>
45              <VOCAB KEY="ALL"
46                  LABEL="All">
47                  <attribute name="ID"/>
48                  <attribute name="_JOIN_KEY"/>
49                  <attribute name="_JOIN_LABEL"/>
50                  <attribute values="Outer Inner"
51                      presence="IMPLIED"
52                      atttype="ENUMERATION"
53                      name="_JOIN_TYPE"
54                      default="Outer"/>
55                  <attribute name="_CASE"/>
56                  <attribute name="_SWITCH"/>
57                  <attribute name="_SORT_BY"/>
58                  <attribute values="YES NO"
59                      presence="IMPLIED"
60                      atttype="ENUMERATION"
61                      name="_CHILDREN_THREADS"
62                      default="YES"/>
63                  <attribute values="YES NO"
64                      presence="IMPLIED"
65                      atttype="ENUMERATION"
66                      name="_SKIP"
67                      default="YES"/>
68                  <attribute values="YES NO"
69                      presence="IMPLIED"
70                      atttype="ENUMERATION"
71                      name="_HIDE_VALUE"

```

FIG. 5

Figure 5. System Parameter File page 2

```

72         default="YES"/>
73     <attribute values="DAO ODBC ADO XML XDF"
74         presence="IMPLIED"
75         atttype="ENUMERATION"
76         name="_DBTYPE"
77         default="ODBC"/>
78     <attribute name="_DBID"/>
79     <attribute values="YES NO XML"
80         presence="IMPLIED"
81         atttype="ENUMERATION"
82         name="_PARSE"
83         default="YES"/>
84     <attribute name="_IMPORT"/>
85     <attribute name="_MAX_ROWS"/>
86     <attribute values="XML ITD XDF TEXT"
87         presence="IMPLIED"
88         atttype="ENUMERATION"
89         name="_IMPORT_TYPE"
90         default="XML"/>
91     <elementType id="BELLEVUE">
92         <any/></elementType>
93     <elementType id="REDMOND">
94         <any/></elementType>
95     <elementType id="SEATTLE">
96         <any/></elementType>
97     <elementType id="FORSALE">
98         <any/></elementType>
99     <elementType id="DBDEFINITION">
100         <string/></elementType>
101     <elementType id="DBJOIN">
102         <any/></elementType>
103     <elementType id="INPUT">
104         <string/></elementType>
105     <elementType id="PROBLEM">
106         <any/></elementType>
107     <elementType id="GENERAL">
108         <any/></elementType>
109     <elementType id="CUSTOMER">
110         <any/></elementType>
111     <elementType id="PROPERTY">
112         <any/></elementType>
113     <elementType id="CONTACT">
114         <any/></elementType>
115     <elementType id="COMPONENT">
116         <any/></elementType>
117     <elementType id="AgentLogout">
118         <any/></elementType>
119     <elementType id="AgentReady">
120         <any/></elementType>
121     <elementType id="AgentNotReady">
122         <any/></elementType>
123     <elementType id="AgentNotBusy">
124         <any/></elementType>
125     <elementType id="Established">
126         <any/></elementType>
127     <elementType id="CallInbound">
128         <any/></elementType>
129     <elementType id="CallOutbound">
130         <any/></elementType>
131     <elementType id="CallWork">
132         <any/></elementType>
133     <elementType id="Released">
134         <any/></elementType>
135     <elementType id="CallHold">
136         <any/></elementType>
137     <ELEMENTS>
138         <DBDEFINITION ICON_INDEX="142"
139             LABEL="Database"/>
140         <INPUT ICON_INDEX="143"
141             LABEL="Input Parameter"/></ELEMENTS></VOCAB>
142     <VOCAB content="CLOSED"

```

FIG. 5

10/28

Figure 5. System Parameter File page 3

```

143         KEY="PROB"
144         LABEL="Problem Log">
145         <attribute name="_JOIN_KEY"/>
146         <attribute name="_JOIN_LABEL"/>
147         <attribute values="Outer Inner"
148             presence="IMPLIED"
149             atttype="ENUMERATION"
150             name="_JOIN_TYPE"
151             default="Outer"/>
152         <attribute name="_CASE"/>
153         <attribute name="_SWITCH"/>
154         <attribute name="_SORT_BY"/>
155         <attribute values="YES NO"
156             presence="IMPLIED"
157             atttype="ENUMERATION"
158             name="_SKIP"
159             default="YES"/>
160         <attribute values="YES NO"
161             presence="IMPLIED"
162             atttype="ENUMERATION"
163             name="_HIDE_VALUE"
164             default="YES"/>
165         <elementType id="DBDEFINITION">
166             <string/></elementType>
167         <elementType id="INPUT">
168             <string/></elementType>
169         <elementType id="PROBLEM">
170             <any/></elementType>
171         <elementType id="LOGOUT">
172             <any/></elementType>
173         <elementType id="READY">
174             <any/></elementType>
175         <elementType id="NOTBUSY">
176             <any/></elementType>
177         <elementType id="NOTREADY">
178             <any/></elementType>
179         <ELEMENTS>
180             <DBDEFINITION ICON_INDEX="142"
181                 LABEL="Database"/>
182             <INPUT ICON_INDEX="143"
183                 LABEL="Input Parameter"/></ELEMENTS></VOCAB>
184     <VOCAB DTD="news.dtd"
185         KEY="SCRIPTINGNEWS"
186         LABEL="ScriptingNews-DTD"/></VOCABULARIES>
187 <ELEMENTS>
188     <BELLEVUE ICON_INDEX="135"
189         LABEL="Bellevue"/>
190     <CUSTOMER ICON_INDEX="138"
191         LABEL="Customer"/>
192     <DBDEFINITION ICON_INDEX="142"
193         LABEL="Database"/>
194     <FORSALE ICON_INDEX="139"
195         LABEL="For Sale"/>
196     <GENERAL ICON_INDEX="145"
197         LABEL="General"/>
198     <INPUT ICON_INDEX="143"
199         LABEL="Input Parameter"/>
200     <DBJOIN ICON_INDEX="144"
201         LABEL="Join Children"/>
202     <PROBLEM ICON_INDEX="147"
203         LABEL="Problem">

```

FIG. 5

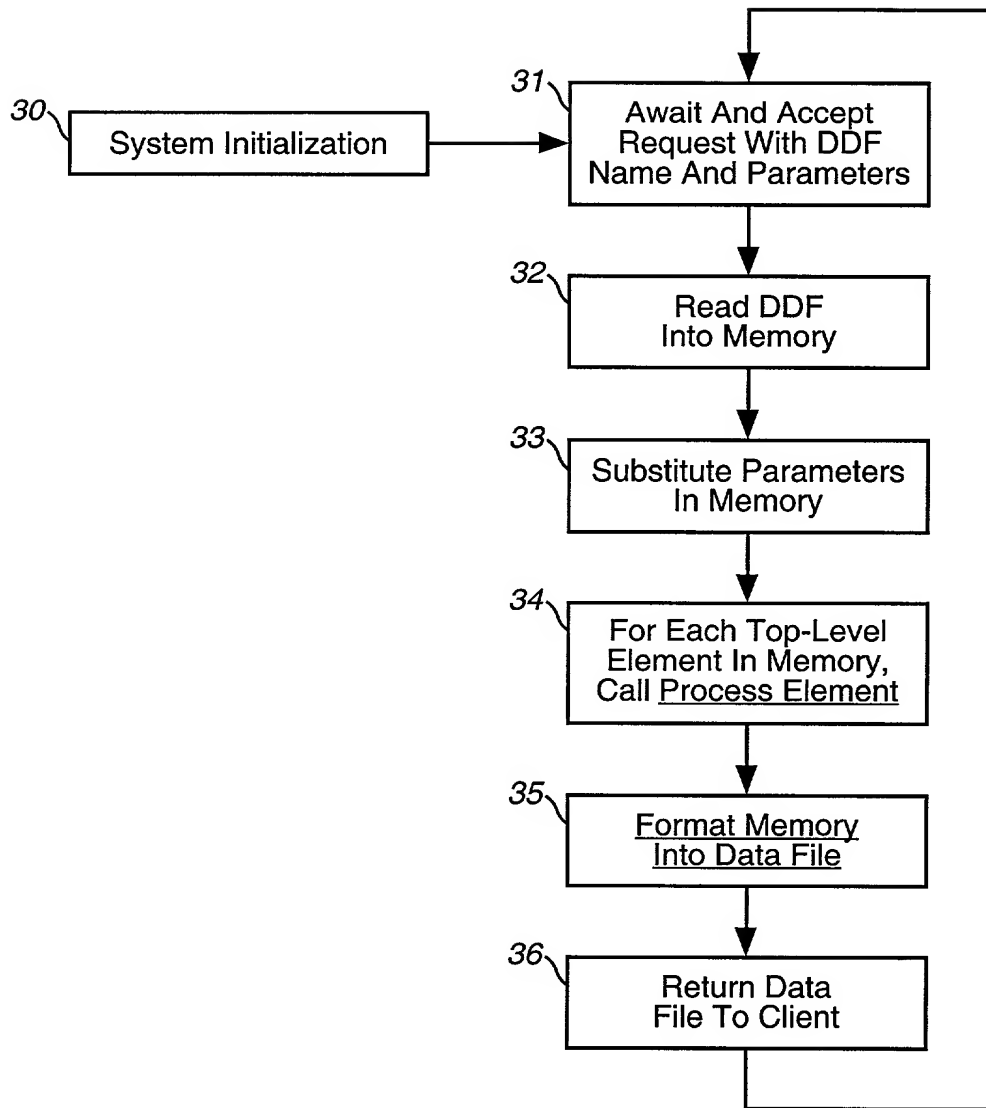
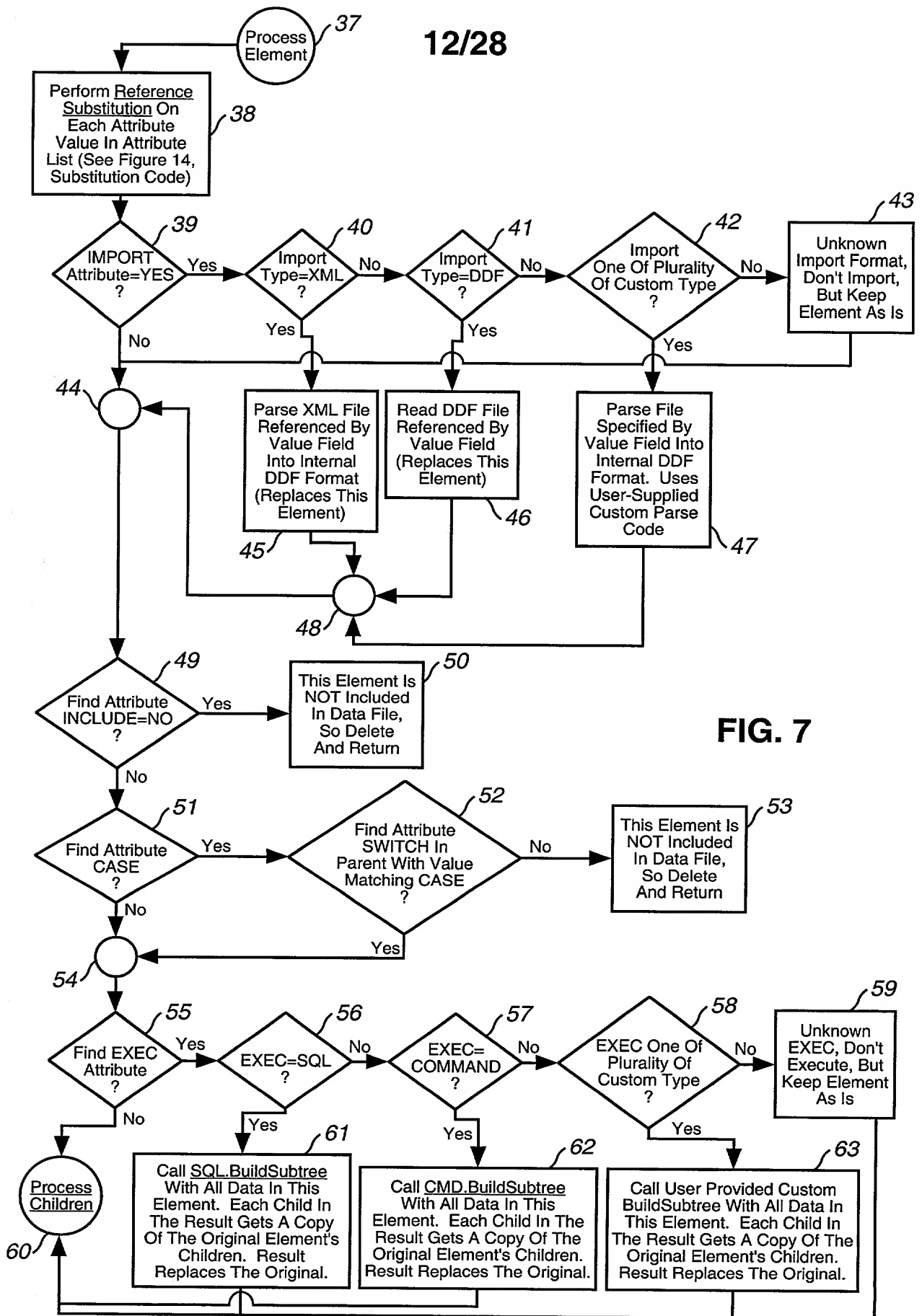


FIG. 6

12/28



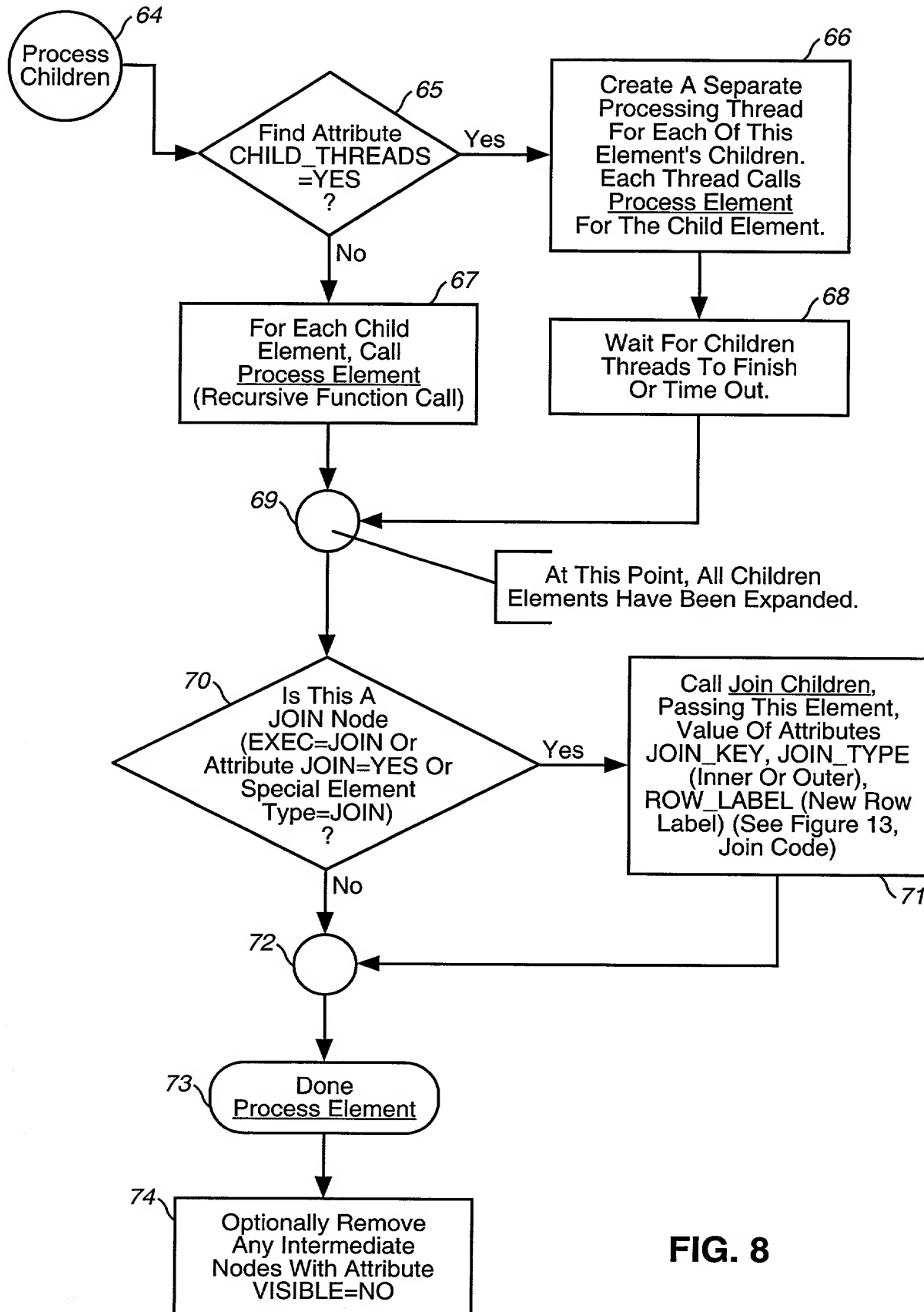


FIG. 8

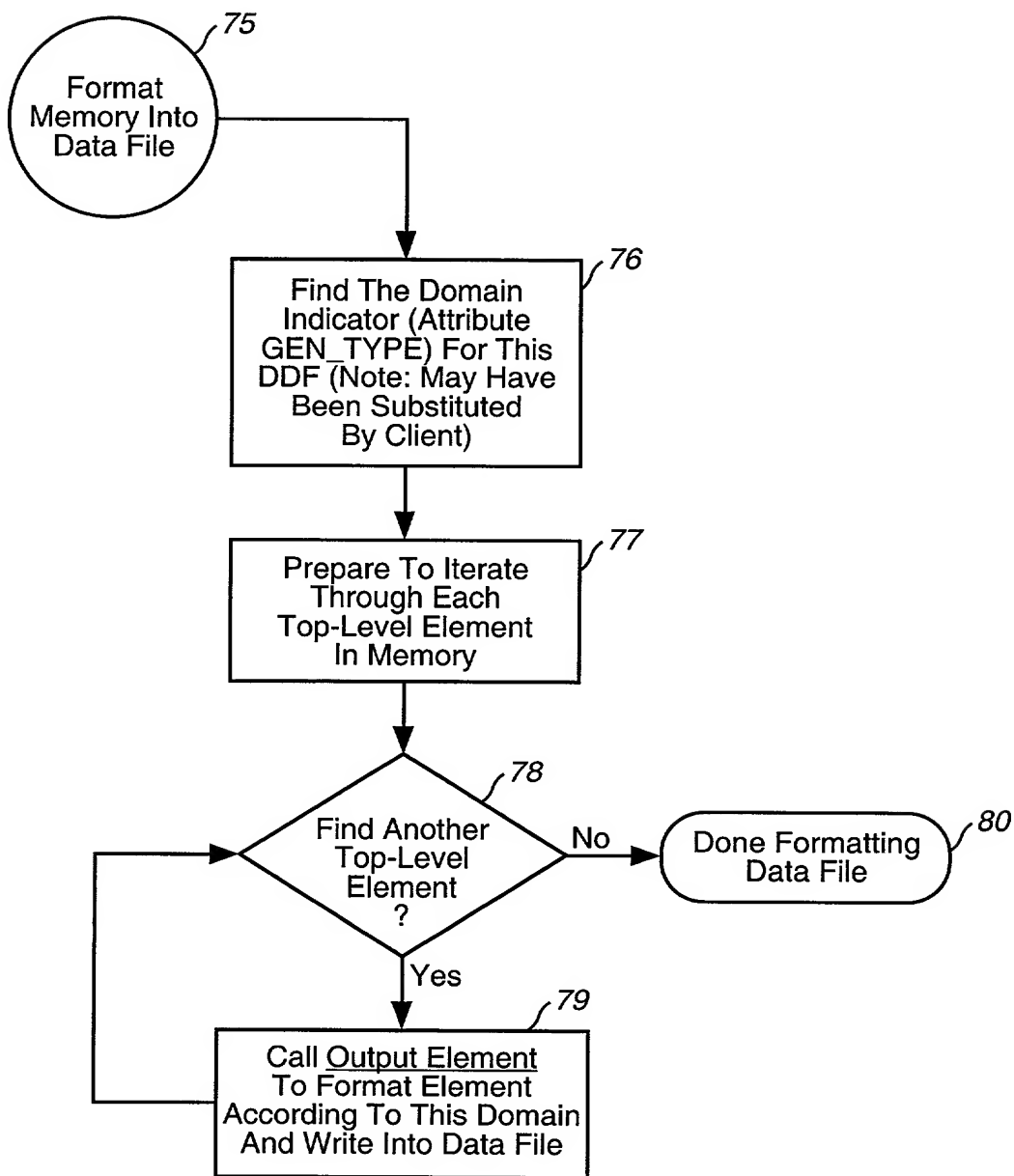
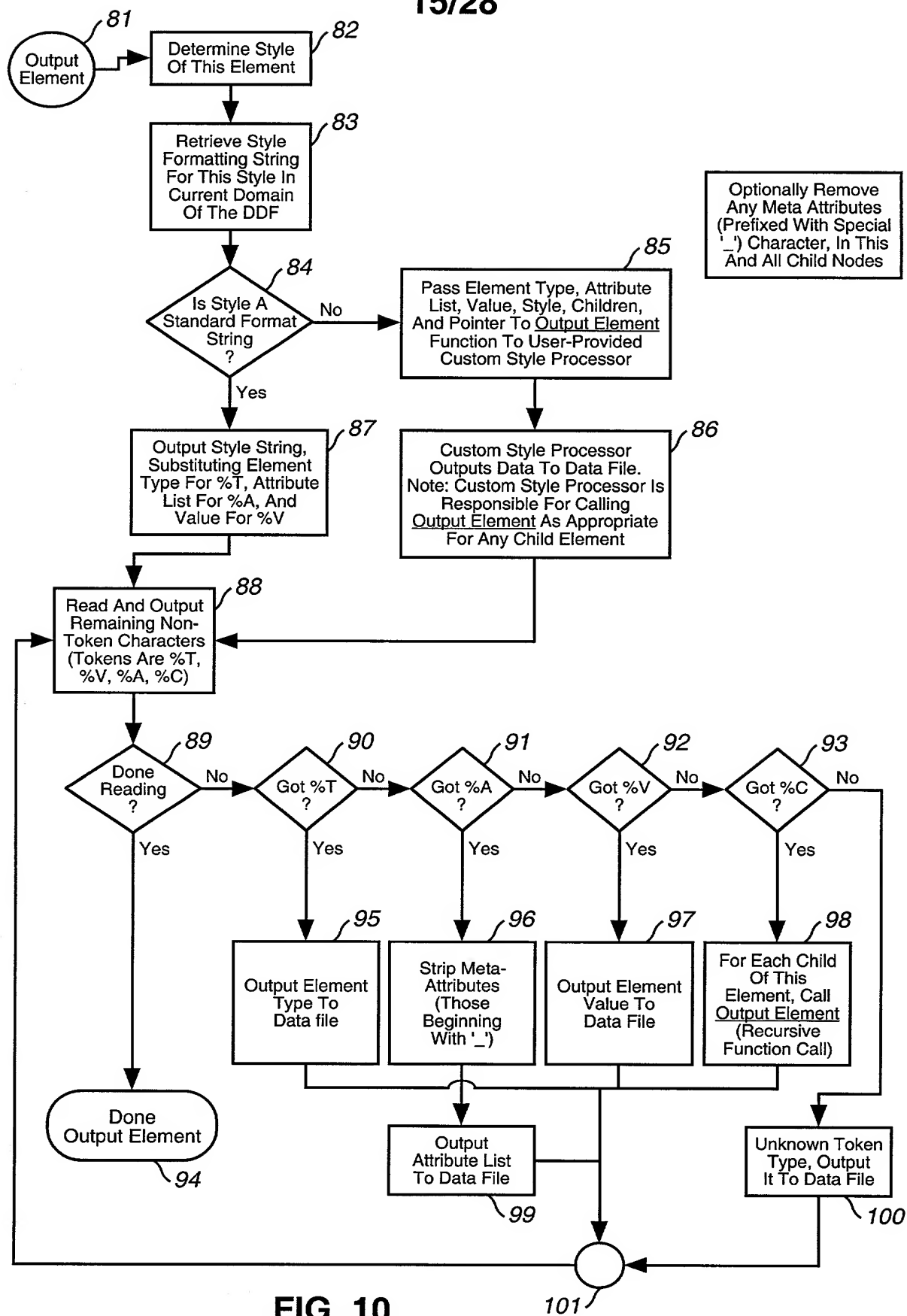


FIG. 9



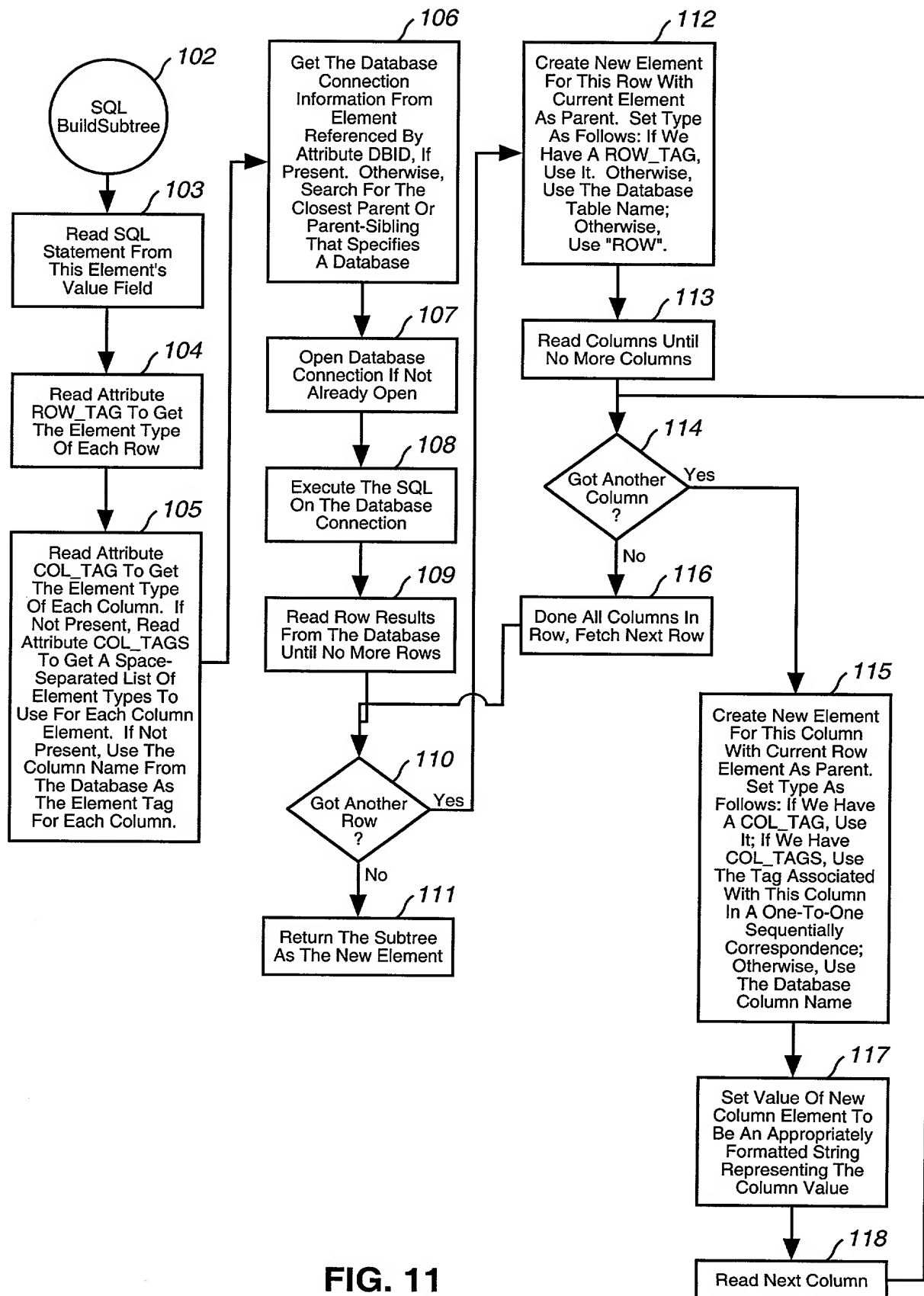


FIG. 11

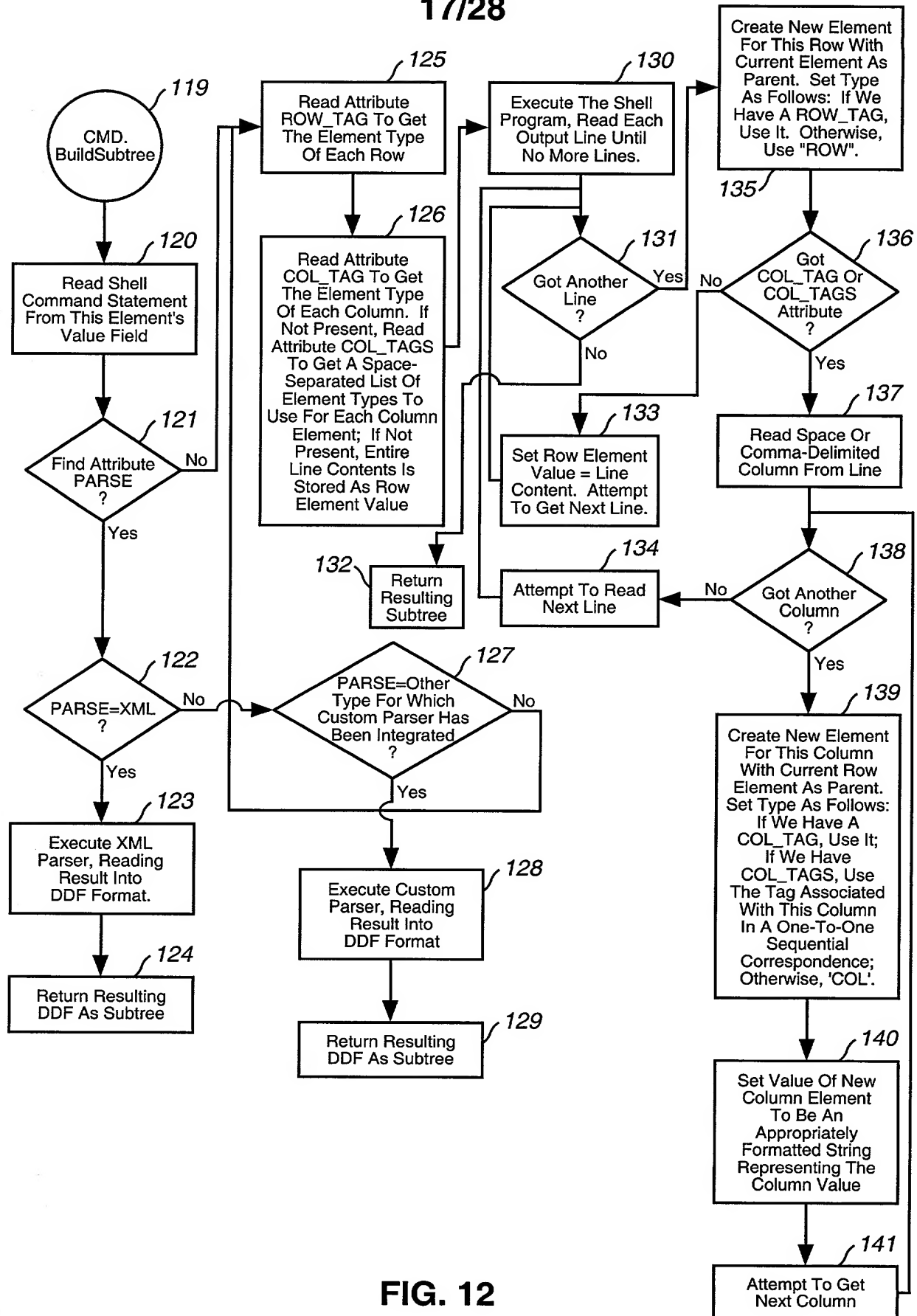


FIG. 12

Figure 13. Join Code page 1

```

// this code called from within Process Element, at the point where we need to check if
JOIN is requested.
// execute any intermediate level nodes
if (sExecType.CompareNoCase(TOKEN_JOIN) == 0) {
    CTreeItem *pBaseItem, *pJoinItem;
    CString sJoin;
    CString sJoinKey;
    CString sJoinLabel;
    JoinType joinType;
    POSITION basePos;

    // parameters for the join:
    // JOIN_TYPE: INNER, OUTER
    // JOIN_KEY: the key used to check the join condition
    // JOIN_LABEL: the new key for the joined row

    // find join parameters
    SearchAttributeValue(CString(TOKEN_JOIN_TYPE), sJoin);
    if (sJoin.CompareNoCase(TOKEN_JOIN_OUTER) == 0)
        joinType = OuterJoin;
    else
        joinType = InnerJoin;

    // get join key
    SearchAttributeValue(CString(TOKEN_JOIN_KEY), sJoinKey);

    // get join label
    SearchAttributeValue(CString(TOKEN_JOIN_LABEL), sJoinLabel);

    // take the first child as the source of the join,
    // join the second
    basePos = m_childList.GetHeadPosition();
    pBaseItem = (CTreeItem *) m_childList.GetNext( basePos );

    while (basePos != NULL) {
        pJoinItem = (CTreeItem *) m_childList.GetNext( basePos );
        pBaseItem = pBaseItem->Join(pJoinItem, sJoinKey, sJoinLabel, joinType);
        ASSERT_VALID(pBaseItem);
    }

    // now prune all children and replace with the new base
    for (pos = m_childList.GetHeadPosition(); (prevPos = pos) != NULL; ) {
        pChildItem = (CTreeItem *) m_childList.GetNext( pos );
        m_childList.RemoveAt( prevPos ); // remove what's at prevPos
        delete pChildItem;
    }

    // add the generated children to this node
    for (pos = pBaseItem->m_childList.GetHeadPosition(); (prevPos = pos) != NULL; ) {
        pChildItem = (CTreeItem *) pBaseItem->m_childList.GetNext(pos);
        pChildItem->m_pParent = this; // reset the parent
        ASSERT_VALID(pChildItem);
        m_childList.AddTail(pChildItem);
        pBaseItem->m_childList.RemoveAt( prevPos ); // remove what's at prevPos
        ASSERT_VALID(this);
    }
} // end TOKEN_JOIN

////////////////////////////////////
// Join - join the input subtree children with the current subtree's children,
// returning a new subtree as the result. The resulting parent is a copy of
// 'this', with joined children
CTreeItem *CTreeItem::Join(CTreeItem *pJoinTree,
                           CString sJoinKey,      // key to join on
                           CString sRowLabel,      // new row label
                           JoinType joinType)
{
    POSITION pos0;
    CTreeItem *pItem0, *pNewSubtree, *pSubtree0, *pSubtree1;

```

FIG. 13

Figure 13. Join Code page 2

```

ASSERT_VALID(pJoinTree);

// operate on the longest list as the source
if (m_childList.GetCount() >= pJoinTree->m_childList.GetCount()) {
    pSubtree0 = this;
    pSubtree1 = pJoinTree;
}
else {
    pSubtree1 = this;
    pSubtree0 = pJoinTree;
}

// create the join subtree to hold results
pNewSubtree = new CTreeItem(m_sKey, m_sValue, m_attrList, m_pParent);

// now do list0 JOIN list1
for (pos0 = pSubtree0->m_childList.GetHeadPosition(); pos0 != NULL; ) {
    CTreeItem *pKeyItem, *pTargetItem, *pMergedItem;
    pItem0 = (CTreeItem *) pSubtree0->m_childList.GetNext(pos0);
    // find the value of the row's Join Key item.
    pKeyItem = pItem0->SearchSubtree(
        (CString) TOKEN_WILDCARD + "/" + sJoinKey, TOKEN_WILDCARD);

    // as long as we find a join key, try and join
    if (pKeyItem) {
        // search the join subtree for "*/*/JOIN_KEY/pKeyItem->m_sValue"
        if ((pTargetItem = pSubtree1->SearchSubtree(
            (CString) TOKEN_WILDCARD + "/"
            + TOKEN_WILDCARD + "/"
            + sJoinKey,
            pKeyItem->m_sValue)) != NULL) {
            // we have a join on this "row"
            CTreeItem *pNewItem = new CTreeItem(*pItem0);
            pNewItem->m_pParent = pNewSubtree; // set the parent
            if (!sRowLabel.IsEmpty())
                pNewItem->m_sKey = sRowLabel;

            // merge for the join. Note that pTargetItem points to the
            // item containing the join key. We want to join at the row
            // level, which is the parent of this node.
            pMergedItem = pNewItem->Merge(pTargetItem->m_pParent);
            delete pNewItem;
            pMergedItem->RemoveDups(); // remove any dup children

            // add the joined child to the new parent
            pMergedItem->m_pParent = pNewSubtree;
            ASSERT_VALID(pMergedItem);
            pNewSubtree->m_childList.AddTail(pMergedItem);
        }
        else if (joinType == OuterJoin) {
            CTreeItem *pNewItem = new CTreeItem(*pItem0);
            pNewItem->m_pParent = pNewSubtree;
            pNewItem->m_sKey = sRowLabel;
            pNewSubtree->m_childList.AddTail(pNewItem);
        }
    }
    else if (joinType == OuterJoin) {
        CTreeItem *pNewItem = new CTreeItem(*pItem0);
        pNewItem->m_pParent = pNewSubtree;
        pNewItem->m_sKey = sRowLabel;
        pNewSubtree->m_childList.AddTail(pNewItem);
    }
}

// return the joined tree
ASSERT_VALID(pNewSubtree);
return pNewSubtree;
}

```

FIG. 13

Figure 13. Join Code page 3

```

////////////////////////////////////
// Merge - Merge 'this' children with the input children, returning a newly
// created subtree
CTreeItem *CTreeItem::Merge(CTreeItem *pMergeTree)
{
    CTreeItem *pTreeItem = new CTreeItem(*this); // copy current tree
    POSITION pos;
    CTreeItem *pChildItem, *pNewChild;

    // copy each subtree to target
    for (pos = pMergeTree->m_childList.GetHeadPosition(); pos != NULL; ) {
        pChildItem = (CTreeItem *) pMergeTree->m_childList.GetNext( pos );
        pNewChild = new CTreeItem(*pChildItem);
        pNewChild->m_pParent = pTreeItem;
        pTreeItem->m_childList.AddTail(pNewChild);
    }

    return(pTreeItem);
}

////////////////////////////////////

// RemoveDups - remove and delete duplicate children. Two children are
// considered dups if they have the same sKey, sValue, and sAttributes
VOID CTreeItem::RemoveDups()
{
    POSITION pos, pos1, prevPos;
    CTreeItem *pTreeItem, *pTargetItem;

    // for each child, remove any dup later in the list
    for (pos = m_childList.GetHeadPosition(); pos != NULL; ) {
        pTargetItem = (CTreeItem *) m_childList.GetNext( pos );

        // see if target is anywhere else in list, remove if so
        pos1 = pos;
        while ((prevPos = pos1) != NULL) {
            pTreeItem = (CTreeItem *) m_childList.GetNext( pos1 );
            if (pTreeItem->m_sKey == pTargetItem->m_sKey
                && pTreeItem->m_sValue == pTargetItem->m_sValue
                && pTreeItem->m_attrList == pTargetItem->m_attrList) {
                m_childList.RemoveAt( prevPos ); // remove at prevPos
                delete pTreeItem;
            }
            pos1 = pos1->GetNext();
        }
        pos = pos->GetNext();
    }
}

////////////////////////////////////
// SearchSubtree - find the first tree item in the subtree given the path
// The subtree path may contain the wildcard character '*', which indicates
// all children of that node should be searched.
// Example: SearchSubtree("*/CUST_ID", "000128") searches all children
// of the current subtree 'this' for the first child with key 'CUST_ID' and
// value '000128'.
CTreeItem *CTreeItem::SearchSubtree(CString sSearchPath, CString sValue)
{
    BOOL pathEnd;
    CString sKey, sRemains;
    int iLoc;
    POSITION pos;
    CTreeItem *pChildItem, *pFound;

    // search all children for the given path component
    iLoc = sSearchPath.Find(SLASH_CHAR);
    if (iLoc >= 0) {
        sKey = sSearchPath.Left(iLoc);
        sRemains = sSearchPath.Mid(iLoc + 1);
        pathEnd = FALSE;
    }
}

```

FIG. 13

Figure 13. Join Code page 4

```

else {
    sKey = sSearchPath;
    pathEnd = TRUE;
}

// if we're at the end of the path, check to see if this node is it.
if (pathEnd) {
    // we have a match if:
    // 1) exact match, 2) sKey is wildcard and value matches,
    // 3) sKey matches and sValue is wildcard, 4) both wildcard
    if ((m_sKey == sKey && m_sValue == sValue)
        || (sKey == TOKEN_WILDCARD && m_sValue == sValue)
        || (m_sKey == sKey && sValue == TOKEN_WILDCARD)
        || (sKey == TOKEN_WILDCARD && sValue == TOKEN_WILDCARD) )
        return this;
}

// for each child, remove any dup later in the list
for (pos = m_childList.GetHeadPosition(); pos != NULL; ) {
    pChildItem = (CTreeItem *) m_childList.GetNext( pos );
    // not the end of path, so keep searching if this is allowable path
    if (sKey == "*" || sKey == pChildItem->m_sKey) {
        if ((pFound = pChildItem->SearchSubtree(sRemains,sValue)) != NULL)
            return pFound;
    }
}

// if we got here and we're at the end of the path, we didn't find it
return NULL;
}

```

FIG. 13

Figure 14. Substitution Code page 1

```

////////////////////////////////////
// find and replace all substitution strings. A substitutable token has the form
// %%REF. %% represents the token prefix, which can be changed by setting the
// parameter TOKEN_PREFIX. REF is an internal document reference of the form
// <complex-path>.<attr>. The "." character is the attribute specifier, and can be
// changed by setting the ATTR_DESIGNATOR system parameter.
// See CTreeItem::GetTreeItemComplexPath for complex-path definition.
// If <complex-path> is omitted, /INPUT.<attr> is assumed, and the token
// evaluates to the attribute value of <attr> in the INPUT element. If a path
// is given with no <attr>, then the token evaluates to the value of the resulting
// element. Some examples: %%FILENAME is transformed to /INPUT.FILENAME, which
// evaluates to the value of the FILENAME attribute in the INPUT element.
// ../ITEM.ID evaluates to the value of the ID attribute in the current parent's
// ITEM element.
CString CTreeItem::Substitution(CString sinput, bool bKeyMap)
{
    CString sRemains = input;
    CString sToken, sResult;
    CString sDef, sValue;
    CTreeItem *pDefItem, *pRoot;
    bool bReplaceSpecial=FALSE;
    _TCHAR cReplace;
    int iLoc=0, iLoc1, len;
    CString sAttr, sPath, sLast;

    len = sRemains.GetLength();
    sResult = input;

    // search for the INPUT element
    pRoot = TreeRoot();

    // parse the input string for replacement tokens
    do {
        sRemains = sResult;
        iLoc = sRemains.Find(TOKEN_PREFIX);
        if (iLoc < 0) // all substitutions performed
            break;

        iLoc += TOKEN_PREFIX_LEN;
        sRemains = sRemains.Mid(iLoc);

        // get end of line or space to end this token
        sToken = sRemains.SpanIncluding(PARAM_CHARS);

        // if this is single name token, put in special-case defaults
        // this is a deprecated feature that will soon go away!
        if (sToken.SpanExcluding(TOKEN_NAME_CHARS).IsEmpty()) {
            sAttr = sToken;
            pDefItem = pRoot->FindChildElement(TOKEN_INPUT);
        }
        else {
            // Split token into path-last-attr parts
            // find the last path component
            if ((iLoc1 = sToken.ReverseFind(SLASH_CHAR)) >= 0) {
                sLast = sToken.Mid(iLoc1); // note: don't discard '/'
                sPath = sToken.Left(iLoc1);
            }
            else {
                sLast = sToken;
                sPath = "";
            }

            // split attr from sLast
            if ((iLoc1 = sLast.ReverseFind(ATTR_DESIGNATOR)) >= 0) {
                sAttr = sLast.Mid(iLoc1+1);
                sLast = sLast.Left(iLoc1);
            }
            else {
                sAttr = "";
            }
        }
    } while (iLoc < len);
}

```

FIG. 14

Figure 14. Substitution Code page 2

```

    }

    // now put the path back together
    sPath = sPath + sLast;

    // get the tree item. If full path given, search from root, otherwise
    // relative path from current tree item
    if (sPath[0] == SLASH_CHAR)
        pDefItem = pRoot->GetTreeItemComplexPath(sPath);
    else
        pDefItem = GetTreeItemComplexPath(sPath);
}

if (pDefItem) {
    // get the substitution value
    if (sAttr.IsEmpty())
        sValue = pDefItem->m_sValue;
    else
        pDefItem->m_attrList.Lookup(sAttr, sValue);
}
else
    sValue = ""; // blank it out

// found it, so make the substitution
// put the sub string into the original
sResult = sResult.Left(iLoc-TOKEN_PREFIX_LEN)
    + sValue
    + sResult.Mid(iLoc + sToken.GetLength());
} while (TRUE);

// perform the character mappings, if requested
if (bKeyMap) {
    // substitute any mapped characters
    CString sMap;
    sMap.Format(XML_CHAR_MAP);

    // do basic error checking
    if (sMap.GetLength()) {
        if (sMap[0] != DEFINE_CHAR && sMap[sMap.GetLength()-1] != DEFINE_CHAR) {
            int iMap;
            len = sRemains.GetLength();
            sRemains = sResult;
            do {
                iMap = sMap.Find(DEFINE_CHAR);
                if (iMap <= 0)
                    break;
                // set flag to replace special chars
                if (sMap[iMap-1] == XML_SPECIAL_REPLACE_FLAG) {
                    bReplaceSpecial = TRUE;
                    cReplace = sMap[iMap+1];
                }
            } else {
                // replace all instances
                do {
                    iLoc = sRemains.Find(sMap[iMap-1]);
                    if (iLoc < 0) // all substitutions performed
                        break;
                    sRemains.SetAt(iLoc, sMap[iMap+1]);
                } while (TRUE);
            }

            sMap = sMap.Mid(iMap+2);
        } while (TRUE);
    }
}

// replace all special chars if indicated
unsigned long ch = XML_SPECIAL_CHAR_VALUE;
if (bReplaceSpecial) {
    for (iLoc = 0; iLoc < sRemains.GetLength(); iLoc++)

```

FIG. 14

Figure 14. Substitution Code page 3

```

        if ((unsigned) sRemains[iLoc] > ch)
            sRemains.SetAt(iLoc, cReplace);
    }

    return sRemains;
}

////////////////////////////////////
// Get the first subtree at the specified complex-path. A complex-path
// has components of form /KEY:ID="VAL1"/SUBKEY:ID="999"/... Only a single
// attribute is supported currently, but this may be expanded in the future
// to support any number of keys. As a shortcut, if attribute name is left out,
// 'ID' is assumed, e.g. above example is same as /KEY:"VAL1"/SUBKEY:"999".
// The path may be relative, using path component '.' to represent the current
// directory, '..' to represent a parent directory. Leading '/' represents
// the root of the tree.
// Return NULL if not found.
CTreeItem *CTreeItem::GetTreeItemComplexPath(CString sPath)
{
    401  BOOL pathEnd;
        CString sKey, sRemains, sIdAttr, sIdValue;
        int iLoc;
        int iIndex, iChild;
        BOOL bAttr=FALSE, bIndex=FALSE;
        POSITION pos;
        CTreeItem *pChildItem, *pFound;

        // check for reference to parent
        if (sPath[0] == DOT_CHAR && sPath[1] == DOT_CHAR) {
            ASSERT_VALID(m_pParent);
            sPath = sPath.Mid(2);
            return m_pParent->GetTreeItemComplexPath(sPath);
        }

        // search all children for the given path component sKey. Strip leading '/', './'
        if (sPath[0] == DOT_CHAR)
            sPath = sPath.Mid(1);
        if (sPath[0] == SLASH_CHAR)
            sPath = sPath.Mid(1);
        iLoc = sPath.Find(SLASH_CHAR);
        if (iLoc >= 0) {
            sKey = sPath.Left(iLoc);
            sRemains = sPath.Mid(iLoc + 1);
            pathEnd = FALSE;
        }
        else {
            sKey = sPath;
            pathEnd = TRUE;
        }

        // within the path component sKey, separate out the identifying attr-value pair(s)
        if ((iLoc = sKey.Find(ATTR_DESIGNATOR)) >= 0) {
            bAttr = TRUE;
            sIdAttr = sKey.Mid(iLoc+1);
            sKey = sKey.Left(iLoc);

            // find attribute value. If we have single value instead of "ID=val" form,
            // then assume the attribute is "ID", and only value is supplied.

            if ((iLoc = sIdAttr.Find(EQUAL_CHAR)) >= 0) {
                sIdValue = sIdAttr.Mid(iLoc+1);
                sIdAttr = sIdAttr.Left(iLoc);
            }
            else {
                sIdValue = sIdAttr;
                sIdAttr = ATTR_ID;
            }
            StripQuotes(sIdValue);
        }
        else if ((iLoc = sKey.Find(INDEX_DESIGNATOR)) >= 0) {

```

FIG. 14

Figure 14. Substitution Code page 4

```

    // hash mark indicates a 1-based index
    CString sIndex;
    bIndex = TRUE;
    sIndex = sKey.Mid(iLoc+1);
    sKey = sKey.Left(iLoc);
    iIndex = atoi(sIndex);
}
else
    bAttr = FALSE;    // no attribute search at this level

//-----
// for each child, search for match and proceed with lower level search
// If children are indexed on ID, lookup the child directly, otherwise, search
// children.
if (m_childLookup) {
    pChildItem = GetHashedChild(sIdValue);    // ID is Hash value

    // not the end of path, so keep searching if this is allowable path
    if (bAttr) {
        if (sKey == pChildItem->m_sKey && pChildItem->m_attrList[sIdAttr] ==
sIdValue) {
            if (pathEnd)
                return pChildItem;
            else if ((pFound = pChildItem->GetTreeItemComplexPath(sRemains)) != NULL)
                return pFound;
        }
    }
    else {    // ignore attributes
        if (sKey == pChildItem->m_sKey) {
            if (pathEnd)
                return pChildItem;
            else if ((pFound = pChildItem->GetTreeItemComplexPath(sRemains)) != NULL)
                return pFound;
        }
    }
}

// otherwise do a sequential search
iChild = 0;
for (pos = m_childList.GetHeadPosition(); pos != NULL; ) {
    pChildItem = (CTreeItem *) m_childList.GetNext( pos );
    // not the end of path, so keep searching if this is allowable path
    if (bAttr) {
        if (sKey == pChildItem->m_sKey && pChildItem->m_attrList[sIdAttr] ==
sIdValue) {
            if (pathEnd)
                return pChildItem;
            else if ((pFound = pChildItem->GetTreeItemComplexPath(sRemains)) != NULL)
                return pFound;
        }
    }
    else if (bIndex) {
        // if this is the right key, increment count and check if we're there
        if (sKey == pChildItem->m_sKey && ++iChild == iIndex) {
            if (pathEnd)
                return pChildItem;
            else if ((pFound = pChildItem->GetTreeItemComplexPath(sRemains)) != NULL)
                return pFound;
        }
    }
    else {    // ignore attributes, get the first of this key
        if (sKey == pChildItem->m_sKey) {
            if (pathEnd)
                return pChildItem;
            else if ((pFound = pChildItem->GetTreeItemComplexPath(sRemains)) != NULL)
                return pFound;
        }
    }
}

// if we got here and we're at the end of the path, we didn't find it

```

FIG. 14

Figure 14. Substitution Code page 5

```
    return NULL;  
}
```

FIG. 14

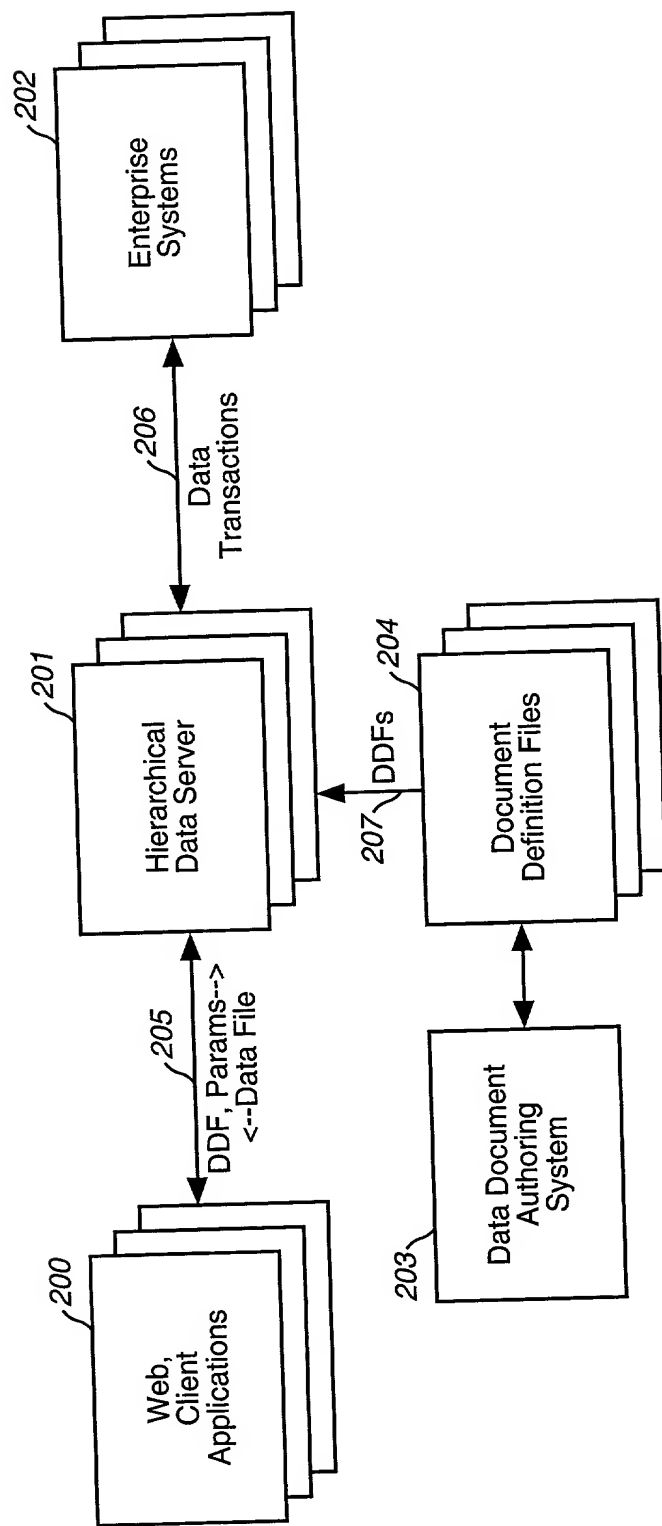


FIG. 15

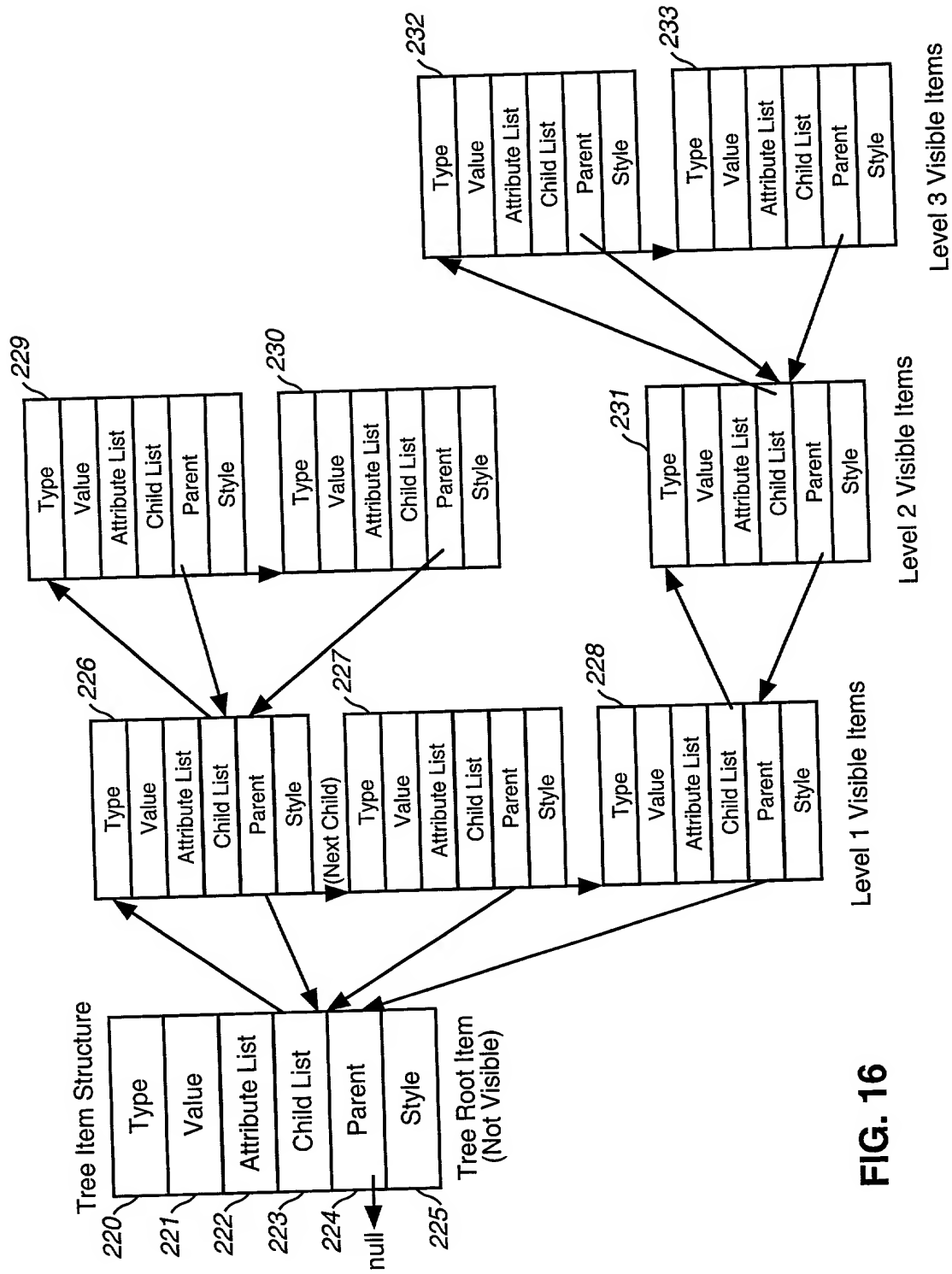


FIG. 16